Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой

работе на тему

СЕТЕВОЙ ЧАТ

БГУИР КР 1-40 02 01 508 ПЗ

Студент Гнетецкий Д.Г.

Руководитель Басак Д.В.

МИНСК 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Язык C++ является одним из самых мощных и универсальных языков программирования на сегодняшний день. Он предоставляет разработчикам широкие возможности для создания сложных и высокопроизводительных приложений. Возможности C++ включают в себя объектно-ориентированное программирование, стандартную библиотеку шаблонов (STL), а также обширные средства для многопоточного и системного программирования. Это делает C++ идеальным выбором для создания многих типов приложений, от игр и десктопных приложений до высокопроизводительных серверных приложений и сложных систем.

В рамках данного курсового проекта разработан сетевой чат с использованием C++ и фреймворка Qt. Несмотря на кажущуюся простоту, процесс разработки сетевого чата требует определённых навыков, таких как обработка событий, работа с графикой и пользовательским интерфейсом, реализация сетевой логики, использование алгоритмов и структур данных. Это требует глубокого понимания принципов программирования и способности применять их на практике.

C++ является основой многих системных и прикладных программ, включая сетевые приложения. Он предлагает разработчикам гибкость и контроль над системными ресурсами, что делает его идеальным выбором для создания высокопроизводительных и надежных приложений. Это особенно важно в сетевых приложениях, где производительность и надежность могут иметь прямое влияние на пользовательский опыт.

C++ и Qt вместе представляют собой мощный инструмент для разработки сетевых приложений. C++ обеспечивает высокую производительность и контроль над системными ресурсами, в то время как Qt предлагает богатый набор функций для работы с графикой, пользовательским интерфейсом и сетевыми протоколами. Это сочетание делает разработку сетевых приложений более эффективной и удобной.

Qt - это кросс-платформенный фреймворк для разработки приложений с графическим интерфейсом пользователя. Он предлагает множество классов и функций, которые упрощают разработку сетевых приложений на C++. Qt поддерживает все основные протоколы и стандарты интернета, включая HTTP, FTP и SSL. Он также предлагает мощные и гибкие средства для работы с графикой и пользовательским интерфейсом. Это делает Qt идеальным выбором для создания современных, красивых и функциональных приложений.

Целью данного проекта является разработка функционального и надежного сетевого чата на C++ с использованием фреймворка Qt. Этот проект предоставит возможность углубить знания в области сетевого программирования, объектно-ориентированного программирования и разработки приложений с графическим интерфейсом пользователя. Это будет

отличной возможностью для практического применения теоретических знаний и улучшения навыков программирования.

Сетевые чаты являются важной частью многих современных приложений и сервисов. Они используются для общения, обмена информацией и совместной работы. Разработка сетевого чата на C++ с использованием Qt позволит не только улучшить навыки программирования, но и получить практический опыт работы с сетевыми технологиями и графическими интерфейсами. Этот опыт будет полезен при разработке других сетевых приложений и сервисов.

Сетевые чаты играют важную роль в современном общении. Они обеспечивают удобный и эффективный способ обмена информацией и идеями между пользователями. Разработка сетевого чата на C++ с использованием Qt - это интересная задача, которая позволяет разработчику улучшить свои навыки и получить ценный опыт в области сетевого программирования.

В заключение, разработка сетевого чата на C++ с использованием Qt - это сложная и интересная задача, которая требует глубоких знаний и навыков. Этот проект будет отличной возможностью для улучшения этих навыков и получения практического опыта в разработке сетевых приложений.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Особенности разработки мессенджеров

Разработка мессенджера - это сложная задача, которая требует учета многих факторов. Во-первых, необходимо обеспечить надежность и скорость передачи сообщений. Это требует использования эффективных алгоритмов и технологий, таких как асинхронное программирование и сетевые протоколы вроде TCP/IP или UDP.

Мессенджер должен быть удобным и простым в использовании. Это означает, что необходимо тщательно продумать интерфейс и обеспечить его интуитивность. Кроме того, мессенджер должен поддерживать различные типы сообщений (текст, изображения, видео, файлы) и предоставлять возможность группового общения.

Чат также должен быть масштабируемым и поддерживать большое количество одновременно подключенных пользователей. Это требует использования технологий, таких как облачные вычисления и распределенные системы.

Все эти аспекты делают разработку мессенджера интересной и сложной задачей, которая позволяет разработчику улучшить свои навыки в различных областях программирования.

1.2Анализ существующих аналогов

Выбор темы курсового проекта был обусловлен несколькими факторами. Во-первых, желанием углубить знания в области языка программирования C++ и объектно-ориентированного программирования. C++ является одним из самых мощных и гибких языков программирования, который используется во многих областях, включая разработку программного обеспечения, системное программирование, разработку игр и многое другое. Объектно-ориентированное программирование, являясь ключевой парадигмой в C++, позволяет создавать модульные и масштабируемые приложения.

Во-вторых, интерес к проектированию пользовательских приложений с использованием популярного фреймворка Qt - это мощный инструмент для создания кросс-платформенных приложений с графическим интерфейсом. Он предлагает широкий спектр функций, включая поддержку сетевого программирования, баз данных, многопоточности и многое другое.

Целью данного проекта не является создание конкурентоспособного продукта на рынке мессенджеров, однако для создания функционального и корректно работающего приложения необходимо провести анализ существующих аналогов. Этот анализ позволит не только получить представление о функционале, который предлагают современные

мессенджеры, но и оценить их преимущества и недостатки, а также учесть эти данные при проектировании собственного приложения.

Изучение рынка мессенджеров позволит получить ценную информацию о последних тенденциях и инновациях в этой области, а также о предпочтениях пользователей. Это поможет создать приложение, которое будет отвечать актуальным требованиям и ожиданиям пользователей. Проведем анализ основных аналогов на рынке мессенджеров, чтобы получить необходимую информацию для дальнейшей работы над проектом.

1.2.1 WhatsApp

WhatsApp — это кроссплатформенный мессенджер, позволяющий обмениваться текстовыми, голосовыми и видеосообщениями, а также файлами различных форматов. Пользователи могут создавать групповые чаты и делиться своим местоположением. WhatsApp использует протоколы шифрования end-to-end, чтобы обеспечить безопасность обмена сообщениями. Однако, в WhatsApp отсутствует возможность управления соединениями со стороны сервера, что является ключевой функцией нашего приложения. На рисунке 1.1 представлен интерфейс WhatsApp:

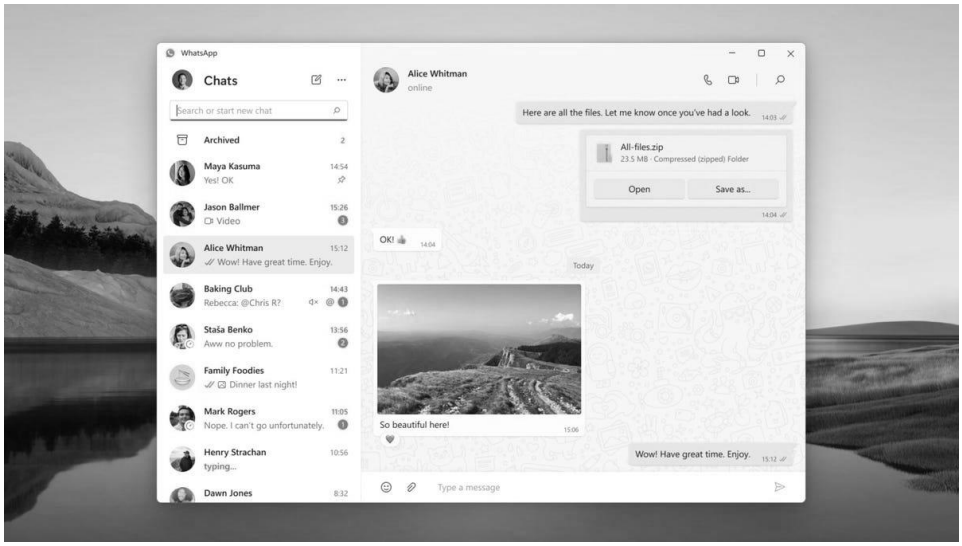

Рисунок 1.1 — Скриншот программы WhatsApp

WhatsApp был разработан компанией WhatsApp Inc. и выпущен в январе 2009 года. Приложение написано на языках Erlang и C++, и поддерживается на всех популярных ОС. WhatsApp активно обновляется и имеет более 2 миллиардов пользователей по всему миру.

1.2.2 Slack

Slack — это мощный корпоративный мессенджер, разработанный с учетом потребностей команд различного размера и специализации. Он предлагает обширный набор функций для обмена текстовыми сообщениями, файлами, а также организации аудио- и видеоконференций.

Slack также предлагает интеграцию с большим количеством сторонних сервисов и приложений, что позволяет автоматизировать многие процессы и упростить работу команды.

Интерфейс Slack разделен на несколько основных областей, включая список каналов и прямых сообщений, область обмена сообщениями и панель управления. На рисунке 1.2 представлен интерфейс Slack.

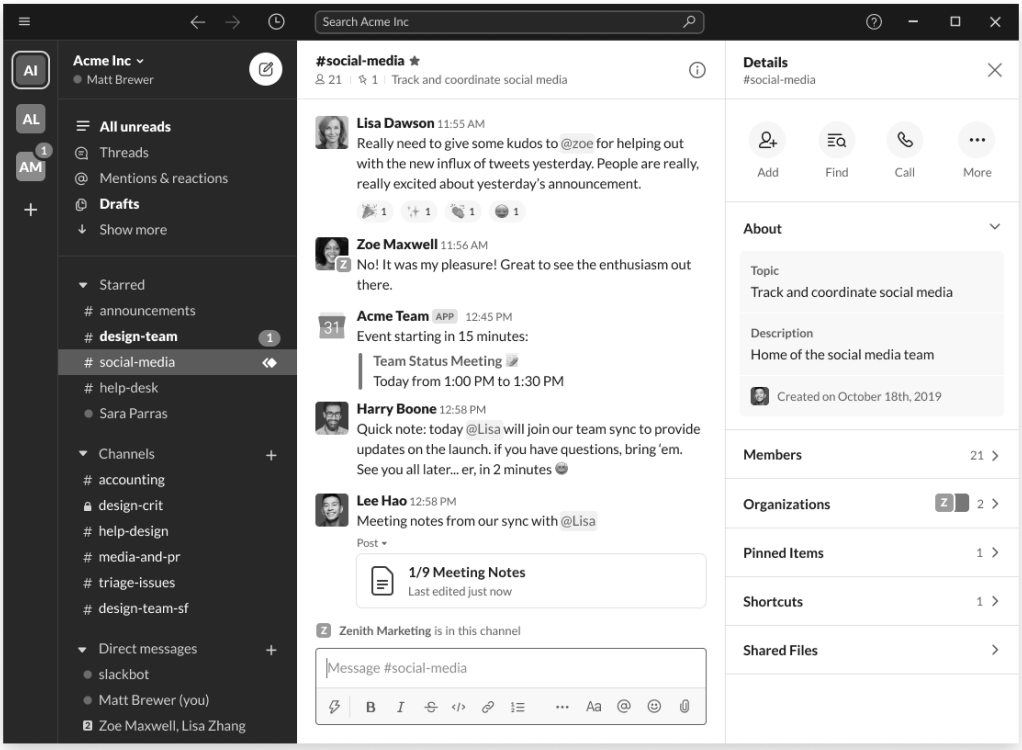

Рисунок 1.2 — Скриншот программы Slack

Slack был разработан компанией Slack Technologies и выпущен в августе 2013 года. Приложение написано на языках Elixir, JavaScript и Objective-C, и поддерживается на всех популярных ОС. Slack активно обновляется и используется многими корпоративными пользователями по всему миру.

1.2.3 Telegram

Telegram — это мультиплатформенный мессенджер, обладающий функциями VoIP. Он предоставляет возможность обмена текстовыми, голосовыми и видеосообщениями, стикерами, фотографиями, а также файлами различных форматов. В Telegram можно осуществлять видео- и аудиозвонки, организовывать конференции, а также создавать многопользовательские группы и каналы.

Telegram использует технологию peer-to-peer для обмена сообщениями, что обеспечивает более быструю передачу информации. Основными преимуществами peer-to-peer являются экономия трафика и повышенное качество связи. Однако есть и недостаток — ваш собеседник может узнать ваш IP-адрес. Интерфейс Telegram представлен на рисунке 1.3.

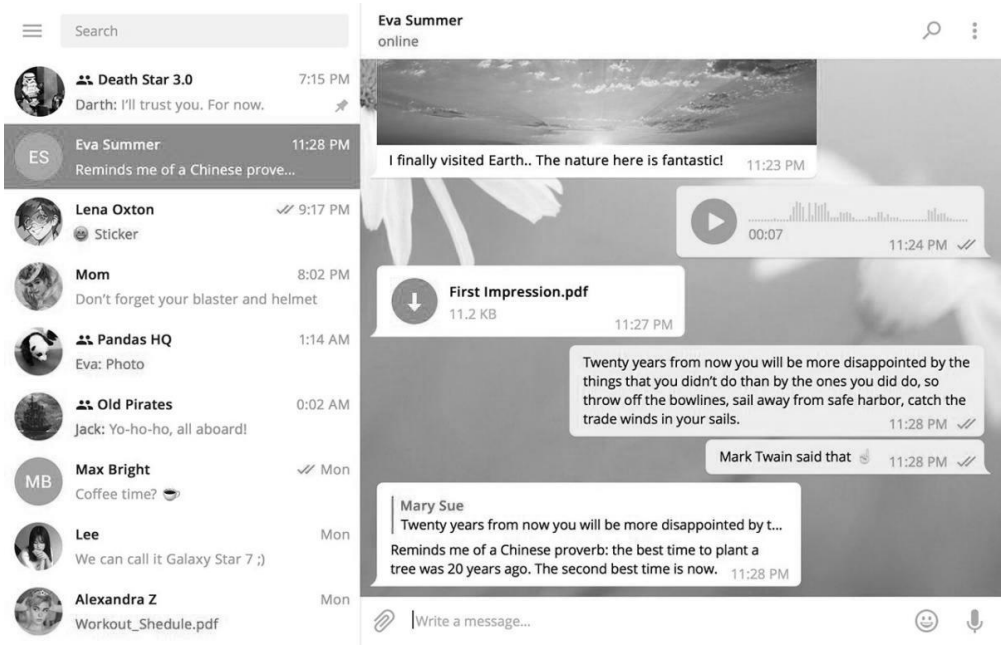

Рисунок 1.3 — Скриншот интерфейса Telegram

Telegram был запущен 14 августа 2013 года. Приложение написано на языках программирования C++ и Java, а пользовательский интерфейс Telegram Desktop реализован с использованием различный фреймворков, включая Qt. Telegram поддерживается на всех распространенных операционных системах и регулярно обновляется.

1.3. Постановка задачи

После анализа существующих сетевых чатов, было решено сосредоточиться на ключевых функциях, которые можно реализовать в рамках данного курсового проекта следующий функционал:

- Разработка серверной части, которая будет управлять соединениями и обменом сообщениями между клиентами.

- Разработка клиентской части, которая будет обеспечивать взаимодействие пользователя с чатом.

- Клиентская часть должна иметь интуитивно понятный пользовательский интерфейс с необходимыми элементами управления.

- В программе должна быть предусмотрена возможность временного хранения истории сообщений.

- Программа должна поддерживать обмен сообщениями в режиме реального времени.

- Программа должна предусматривать возможность авторизации пользователей.

В качестве языка программирования выбран C++, так как он обеспечивает высокую производительность, необходимую для обработки большого количества сообщений, поддерживает объектно-ориентированный подход к программированию и уже знаком по предыдущему опыту.

Для реализации графического интерфейса клиентской части будет использован фреймворк Qt, основанный на языке C++. Qt предлагает обширный набор базовых классов для создания пользовательских классов графического интерфейса, удобную систему взаимодействия между виджетами приложения с помощью системы сигналов и слотов, а также хорошую документацию, что облегчает изучение и использование Qt.

Выбранный набор инструментов позволяет реализовать все поставленные задачи в рамках курсового проекта.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

2.1 Модуль авторизации

Модуль авторизации играет ключевую роль в обеспечении безопасности и индивидуальности пользовательского опыта в чате. Этот модуль отвечает за сбор и проверку уникальности имени пользователя. При входе в систему пользователь вводит свое имя, которое затем проверяется на уникальность среди уже подключенных пользователей. Это обеспечивает, что каждый пользователь в чате имеет уникальное имя, что упрощает идентификацию и общение. Для проверки уникальности имени используется база данных, что обеспечивает надежность и эффективность процесса.

2.2 Модуль пользовательского интерфейса

Модуль пользовательского интерфейса является центральным элементом взаимодействия пользователя с приложением. Он обеспечивает функционал отправки и получения сообщений, отображает список активных пользователей в чате и позволяет выбирать пользователя, которому предназначено сообщение. Этот модуль также отвечает за отображение истории сообщений, что позволяет пользователям просматривать предыдущие обсуждения и участвовать в них. Ввод пользователя также обрабатывается в этом модуле, что обеспечивает гладкое и интуитивно понятное взаимодействие.

2.3 Модуль сервера

Модуль сервера является основой всего приложения. Он слушает входящие подключения, создает новую вкладку для каждого подключенного клиента и обрабатывает обмен сообщениями между клиентами. Сервер также имеет возможность отправлять собственные сообщения каждому клиенту, что позволяет обеспечивать обратную связь и управлять взаимодействием. Кроме того, сервер отслеживает активность пользователей и управляет их соединениями, включая возможность принудительного отключения в случае необходимости. В случае возникновения ошибок, сервер обрабатывает их и предоставляет обратную связь пользователю, что обеспечивает стабильность и надежность работы приложения.

2.4 Модуль протокола общения

Модуль протокола общения определяет структуру и формат данных, используемых для обмена сообщениями между сервером и клиентами. Он отвечает за сериализацию и десериализацию данных, что обеспечивает их корректное представление и обработку. Этот модуль также определяет

различные типы сообщений, такие как отправка текста, отправка имени пользователя, индикатор ввода и другие. Это позволяет обеспечить гибкость и многофункциональность общения в чате. Модуль протокола общения также отвечает за обработку входящих сообщений, извлечение данных из них и предоставление этих данных другим частям приложения для дальнейшей обработки.

2.5 Модуль чтения и записи данных

Модуль чтения и записи данных играет важную роль в обработке входящих и исходящих сообщений. Он считывает введенные пользователем сообщения и отправляет их другим пользователям, обеспечивая тем самым обмен информацией в чате. Кроме того, этот модуль обрабатывает входящие сообщения и добавляет их в историю чата, что позволяет пользователям просматривать предыдущие обсуждения и участвовать в них. Этот модуль обеспечивает эффективное и надежное чтение и запись данных, что является ключевым для функционирования приложения.

2.6 Модуль временного хранения данных

Модуль временного хранения данных служит для организации промежуточного сохранения информации в процессе работы с приложением обмена сообщениями. Данные о текущем пользователе, взаимодействующем с приложением, а также полученные и отправленные сообщения, должны быть временно сохранены для обеспечения непрерывной работы приложения.

Структурная схема представлена в приложении В.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого приложения.

3.1 Разработка диаграммы классов

Диаграмма классов клиента и сервера показана в приложениях А и Б.

3.2 Описание классов клиента

3.2.1 ChatMessageInfo

Класс наследуется от библиотечного класса `QWidget`. Данный класс используется для отображения информации о сообщении в чате.

Поля:

– `Ui::ChatMessageInfo *ui` – указатель на объект, представляющий пользовательский интерфейс для этого виджета.

– `static QMap<QString, QColor> userColorMap` – статическая карта, которая хранит соответствие между именами пользователей и цветами, используемыми для отображения их сообщений.

Методы:

– `ChatMessageInfo` – конструктор, принимает `QWidget*` – ссылку на родительский виджет.

– `~ChatMessageInfo` – деструктор, удаляет объект ui.

– `void displayMessage` – принимает `const QString&` – сообщение, `const QString&` – имя пользователя, `bool` – флаг, указывающий, является ли сообщение сообщением от пользователя. Отображает сообщение, устанавливает его цвет и выравнивание.

– `QColor getUserColor` – принимает `const QString&` – имя пользователя. Возвращает цвет, который должен быть использован для отображения его сообщений.

– `void setMessageAlignment` – принимает `bool` – флаг, указывающий, является ли сообщение сообщением от пользователя. Устанавливает выравнивание сообщения в соответствии с этим флагом.

– `void setMessageColor` – принимает `const QColor&` – цвет. Устанавливает его как цвет текста сообщения.

3.2.2 ClientManager

Класс наследуется от `QObject` и используется для управления взаимодействием клиента с сервером.

Поля:

— `QTcpSocket *socket` — сокет для подключения к серверу.

— `QHostAddress ip` — IP-адрес сервера.

— `ushort port` — порт сервера.

— `ClientProtocol protocol` — объект протокола клиента для обработки сообщений.

Методы:

— `ClientManager` — конструктор, принимает `QHostAddress` — IP-адрес сервера, `ushort` — порт сервера, `QObject*` — ссылку на родительский объект.

— `void connectToServer` — подключается к серверу.

— `void composeAndSendMessage` — принимает `QString` — сообщение, `QString` — получатель. Составляет и отправляет сообщение.

— `void composeAndSendName` — принимает `QString` — имя. Составляет и отправляет сообщение с именем.

— `void disconnectFromServer` — отключается от сервера.

— `void readyRead` — обрабатывает входящие данные от сервера.

— `void setupClient` — настраивает клиентский сокет и соединения.

Сигналы:

— `void connected` — сигнал о подключении к серверу.

— `void disconnected` — сигнал об отключении от сервера.

— `void chatMessageReceived` — сигнал о получении сообщения в чате.

— `void newNameReceived` — сигнал о получении нового имени.

— `void connectionAcknowledged` — сигнал о подтверждении подключения.

— `void newClientConnectedToServer` — сигнал о подключении нового клиента к серверу.

— `void clientNameUpdated` — сигнал об обновлении имени клиента.

— `void clientDisconnected` — сигнал об отключении клиента.

— `void errorOccurred` — сигнал о возникновении ошибки.

3.2.3 ClientProtocol

Класс используется для обработки сообщений между клиентом и сервером.

Поля:
— `MessageType messageType` — тип сообщения.
— `QString chatMessage` — текст сообщения чата.
— `QString newName` — новое имя.
— `QString messageReceiver` — получатель сообщения.
— `QString currentClientName` — текущее имя клиента.
— `QString previousName` — предыдущее имя.
— `QStringList clientNames` — список имен клиентов.
— `QString myName` — мое имя.
— `QString messageSender` — отправитель сообщения.

Методы:
— `ClientProtocol` — конструктор.
— `QByteArray composeTextMessage` — принимает `QString` — сообщение, `QString` — получатель. Составляет и возвращает сообщение чата.
— `QByteArray composeNameMessage` — принимает `QString` — имя. Составляет и возвращает сообщение с именем.
— `void parseData` — принимает `QByteArray` — данные. Разбирает данные.
— `const QString &getChatMessage` — возвращает текст сообщения.
— `const QString &getNewName` — возвращает новое имя.
— `MessageType getMessageType` — возвращает тип сообщения.
— `QString getMessageReceiver` — возвращает получателя сообщения.
— `const QString &getCurrentClientName` — возвращает текущее имя клиента.
— `const QString &getPreviousName` — возвращает предыдущее имя.
— `const QStringList &getClientNames` — возвращает список имен клиентов.
— `const QString &getMyName` — возвращает мое имя.
— `QString getMessageSender` — возвращает отправителя сообщения.
— `QByteArray prepareData` — принимает `MessageType` — тип, `QString` — данные. Подготавливает и возвращает данные.

3.2.4 ClientWindow

Класс представляет собой главное окно клиента.

Поля:
— `Ui::ClientWindow *ui` — интерфейс пользователя.
— `ClientManager *client` — менеджер клиента.
— `LoginWindow *loginWindow` — окно входа.

Методы:
— `ClientWindow` — конструктор.
— `~ClientWindow` — деструктор.
— `void connectToServer` — подключается к серверу.
— `void setupClient` — настраивает клиента.
— `void createMessage` — принимает `QString` — имя пользователя, `QString` — сообщение, `bool` — является ли сообщение моим. Создает сообщение.
— `void processMessageAndSend` — обрабатывает и отправляет сообщение.
— `void on_btnSend_clicked` — обработчик нажатия кнопки "Отправить".
— `void receiveChatMessage` — принимает `QString` — отправитель, `QString` — сообщение. Получает сообщение чата.
— `void onConnectionAcknowledgement` — принимает `QString` — мое имя, `QStringList` — имена клиентов. Обрабатывает подтверждение подключения.
— `void onNewClientConnectedToServer` — принимает `QString` — имя клиента. Обрабатывает подключение нового клиента к серверу.
— `void onClientNameUpdated` — принимает `QString` — предыдущее имя, `QString` — имя клиента. Обрабатывает обновление имени клиента.
— `void onClientDisconnected` — принимает `QString` — имя клиента. Обрабатывает отключение клиента.
— `void on_nameEdit_returnPressed` — обработчик нажатия кнопки "Ввод" в поле редактирования имени.
— `void on_editMessage_returnPressed` — обработчик нажатия кнопки "Ввод" в поле редактирования сообщения.

3.2.5 LoginWindow

Класс представляет собой окно входа.

Поля:
— `Ui::LoginWindow *ui` — интерфейс пользователя.
— `QSqlDatabase db` — база данных.
— `QString nickName` — никнейм.

Методы:
— `LoginWindow` — конструктор.
— `~LoginWindow` — деструктор.
— `QString getNickname` — возвращает никнейм.
— `void removeNickname` — принимает `const QString &nickname`. Удаляет никнейм.
— `bool doesNicknameExist` — принимает `const QString &nickname`. Проверяет, существует ли никнейм.
— `void addNickname` — принимает `const QString &nickname`. Добавляет никнейм.
— void `setNickName` — принимает `const QString &newNickName`. Устанавливает никнейм.
— `void setupDatabase` — настраивает базу данных.
— `void on_nickname_returnPressed` — обработчик нажатия кнопки "Ввод" в поле никнейма.

3.3 Описание классов сервера

3.3.1 ChatWindow

Класс представляет собой окно чата.

Поля:
— `Ui::ChatWindow *ui` — интерфейс пользователя.
— `ServerClientManager *client` — менеджер клиента сервера.

Методы:
— `ChatWindow` — конструктор.
— `~ChatWindow` — деструктор.
— `void disconnect` — отключается от сервера.
— `void clientDisconnected` — обрабатывает отключение клиента.
— `void on_btnSend_clicked` — обработчик нажатия кнопки "Отправить".

— void chatMessageReceived — принимает QString — сообщение, QString — получатель, QString — отправитель. Обрабатывает получение сообщения чата.

— void onClientNameChanged — принимает QString — предыдущее имя, QString — имя. Обрабатывает изменение имени клиента.

Сигналы:

— void clientNameUpdated — принимает QString — предыдущее имя, QString — новое имя. Вызывается при обновлении имени клиента.

— void textForOtherClients — принимает QString — сообщение, QString — получатель, QString — отправитель. Вызывается при отправке текста другим клиентам.

3.3.2 ServerClientManager

Класс представляет собой менеджер клиента на стороне сервера.

Поля:

— QTcpSocket *socket — сокет.

— QHostAddress ip — IP-адрес.

— ushort port — порт.

— ServerProtocol protocol — протокол сервера.

Методы:

— ServerClientManager — конструкторы.

— void connectToServer — подключается к серверу.

— void disconnectFromServer — отключается от сервера.

— void composeAndSendMessage — принимает QString — сообщение. Компонует и отправляет сообщение.

— void composeAndSendName — принимает QString — имя. Компонует и отправляет имя.

— QString getClientName — возвращает имя клиента.

— void readyRead — обрабатывает данные из сокета.

— void setupClient — настраивает клиента.

Сигналы:

— void connected — вызывается при подключении.

— void disconnected — вызывается при отключении.

— void chatMessageReceived — принимает const QString — сообщение, QString — получатель, QString — отправитель. Вызывается при получении сообщения чата.

— `void clientNameUpdated` — принимает `QString` — предыдущее имя, `QString` — новое имя. Вызывается при обновлении имени клиента.

3.3.3 ServerManager

Класс представляет собой менеджер сервера.

Поля:
— `QTcpServer *server` — сервер.
— `ServerProtocol protocol` — протокол сервера.
— `QMap<QString, QTcpSocket *> clients` — карта клиентов.

Методы:
— `ServerManager` — конструктор.
— `void informClientsAboutNameChange` — принимает `QString` — предыдущее имя, `QString` — новое имя. Информирует клиентов об изменении имени.
— `void onTextForOtherClients` — принимает `QString` — сообщение, `QString` — получатель, `QString` — отправитель. Обрабатывает текст для других клиентов.
— `void onNewClientConnection` — обрабатывает новое подключение клиента.
— `void onClientDisconnected` — обрабатывает отключение клиента.
— `void setupServer` — принимает `ushort` — порт. Настраивает сервер.

Сигналы:
— `void newClientConnected` — принимает `QTcpSocket *client`. Вызывается при подключении нового клиента.
— `void clientDisconnected` — принимает `QTcpSocket *client`. Вызывается при отключении клиента.

3.3.4 ServerProtocol

Класс представляет собой протокол сервера.

Поля:
— `MessageType messageType` — тип сообщения.
— `QString chatMessage` — сообщение чата.

— QString newName — новое имя.
— QString messageReceiver — получатель сообщения.

Методы:
— ServerProtocol — конструктор.
— QByteArray composeChatMessage — принимает QString — сообщение, QString — получатель, QString — отправитель. Компонует сообщение чата.
— QByteArray composeNameMessage — принимает QString — имя. Компонует сообщение имени.
— QByteArray composeUpdateNameMessage — принимает QString — предыдущее имя, QString — имя. Компонует сообщение обновления имени.
— QByteArray composeConnectionAckMessage — принимает QString — имя клиента, QStringList — другие клиенты. Компонует сообщение подтверждения подключения.
— QByteArray composeNewClientMessage — принимает QString — имя клиента. Компонует сообщение нового клиента.
— QByteArray composeClientDisconnectedMessage — принимает QString — имя клиента. Компонует сообщение об отключении клиента.
— void parseData — принимает QByteArray — данные. Разбирает данные.
— const QString &getChatMessage — возвращает сообщение чата.
— const QString &getNewName — возвращает новое имя.
— MessageType getMessageType — возвращает тип сообщения.
— const QString &getMessageReceiver — возвращает получателя сообщения.
— QByteArray getData — принимает MessageType — тип, QString — данные. Возвращает данные.

3.3.5 ServerWindow

Класс представляет собой окно сервера.

Поля:
— Ui::ServerWindow *ui — интерфейс пользователя.
— ServerManager * server — менеджер сервера.

Методы:
— ServerWindow — конструктор.

— ~ServerWindow — деструктор.

— void newClientConnected — принимает QTcpSocket *client. Обрабатывает подключение нового клиента.

— void clientDisconnected — принимает QTcpSocket *client. Обрабатывает отключение клиента.

— void updateClientName — принимает QString — предыдущее имя, QString — имя. Обновляет имя клиента.

— void on_tabChats_tabCloseRequested — принимает int — индекс. Обрабатывает запрос на закрытие вкладки чата.

— void setupServerConfiguration — настраивает конфигурацию сервера.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка алгоритмов

Метод informClientsAboutNameChange(QString prevName, QString name) класса ServerManager.

Шаг 1. Начало. Этот метод вызывается, когда клиент меняет свое имя.

Шаг 2. Формируем сообщение о смене имени, включающее предыдущее и новое имя. Это сообщение будет отправлено всем клиентам.

Шаг 3. Проходим по всему списку подключенных клиентов и выполняем шаги 4-6, после чего переходим к шагу 7. Это необходимо, чтобы уведомить всех клиентов о смене имени.

Шаг 4. Получаем имя текущего клиента. Это имя используется для проверки, является ли текущий клиент тем, кто меняет имя.

Шаг 5. Если имя текущего клиента не равно новому имени, переходим к шагу 6, иначе переходим к шагу 3. Это предотвращает отправку сообщения клиенту, который меняет имя.

Шаг 6. Отправляем текущему клиенту сообщение о смене имени. Это уведомляет клиента о том, что один из других клиентов изменил имя.

Шаг 7. Конец. Завершаем метод после уведомления всех клиентов.

Метод processMessageAndSend() класса ClientWindow.

Шаг 1. Начало. Этот метод вызывается, когда клиент хочет отправить сообщение.

Шаг 2. Получаем текст из поля ввода сообщения и удаляем пробелы с обеих сторон. Это гарантирует, что сообщение не будет начинаться или заканчиваться пробелами.

Шаг 3. Если текст пуст, то переходим к шагу 4, если нет, то переходим к шагу 5. Это предотвращает отправку пустых сообщений.

Шаг 4. Выводим предупреждение о том, что сообщение не может быть пустым и выходим из метода. Это информирует пользователя о том, что сообщение должно содержать текст.

Шаг 5. Компонуем и отправляем сообщение, используя введенный текст и выбранного в комбо-боксе получателя. Это гарантирует, что сообщение будет отправлено правильному получателю.

Шаг 6. Очищаем поле ввода сообщения. Это готовит поле ввода для следующего сообщения.

Шаг 7. Создаем сообщение с введенным текстом и флагом, указывающим, что сообщение отправлено. Это отображает отправленное сообщение в интерфейсе пользователя.

Шаг 8. Конец. Завершаем метод после отправки сообщения.

Метод on_nameEdit_returnPressed() класса ClientWindow.

Шаг 1. Начало. Этот метод вызывается, когда пользователь нажимает клавишу возврата в поле ввода имени.

Шаг 2. Получаем новое имя из соответствующего поля ввода и удаляем пробелы с обеих сторон. Это гарантирует, что имя не будет начинаться или заканчиваться пробелами.

Шаг 3. Если новое имя уже существует в базе данных, переходим к шагу 4, иначе переходим к шагу 5. Это предотвращает использование уже занятого имени.

Шаг 4. Выводим предупреждение о том, что такое имя уже занято, устанавливаем в поле ввода текущее имя и выходим из метода. Это информирует пользователя о том, что выбранное имя уже используется.

Шаг 5. Удаляем текущее имя из базы данных. Это освобождает имя для использования другими пользователями.

Шаг 6. Добавляем новое имя в базу данных. Это регистрирует новое имя в системе.

Шаг 7. Устанавливаем новое имя как текущее. Это обновляет имя пользователя в интерфейсе.

Шаг 8. Компонуем и отправляем сообщение с новым именем. Это уведомляет сервер о смене имени.

Шаг 9. Конец. Завершаем метод после смены имени.

4.2 Исходный текст программы

Код программы представлен в приложении Г.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЯ

Тестирование является критическим этапом в процессе разработки программного обеспечения. Это процесс, в котором мы анализируем функциональность приложения или системы в контролируемых условиях. Создание различных сценариев и условий позволяет нам увидеть, как приложение ведет себя в различных ситуациях, и это помогает нам определить, работает ли оно так, как предполагалось.

Важность тестирования не может быть недооценена. Оно предоставляет нам ценные данные, которые мы можем использовать для планирования будущих улучшений и развития приложения. Тестирование действует как форма диагностики, выявляя проблемы и недостатки, которые могут повлиять на дальнейшие шаги в процессе разработки.

В процессе тестирования разработанного приложения необходимо сымитировать различные ошибки и посмотреть реакцию приложения на них. Это позволяет увидеть, как приложение справляется с непредвиденными обстоятельствами и проблемами, и дает возможность улучшить его устойчивость и надежность.

5.1 Тестирование работы при отсутствии подключения к серверу

В данном тесте проверяется, как приложение реагирует на отсутствие подключения к серверу. Это важно, поскольку в реальных условиях сервер может быть недоступен по различным причинам - от сбоев в сети до технического обслуживания сервера.

На представленном скриншоте (рисунок 5.1.1) видно, что при попытке подключения к серверу, который в данный момент не запущен, приложение выдает ошибку. Это ожидаемое поведение, поскольку приложение не может установить соединение с сервером.

Рисунок 5.1.1 — Сервер не запущен.

Важно, чтобы приложение корректно обрабатывало такую ситуацию и информировало пользователя о проблеме, а не просто "зависало" или выдавало неинформативные сообщения об ошибках.

5.2 Тестирование функциональности входа с уникальным именем пользователя

В данном тесте проверяется, как приложение обрабатывает ситуацию, когда новый пользователь пытается войти в чат с именем, которое уже занято другим пользователем. Это важно для поддержания уникальности идентификаторов пользователей и предотвращения путаницы в чате.

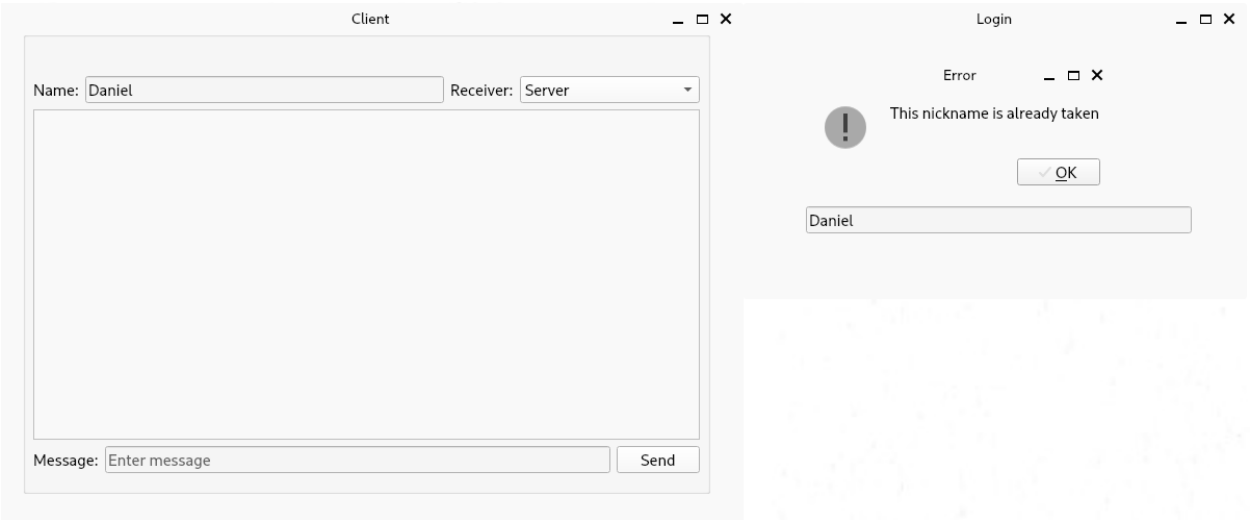

Рисунок 5.1.2 — Попытка входа с именем, которое уже занято.

На представленном скриншоте (рисунок 5.1.2) видно, что один пользователь уже вошел в чат под именем "Daniel". Когда другой пользователь пытается войти в чат с тем же именем, приложение корректно выдает предупреждение о том, что имя уже занято. Это подтверждает, что функциональность проверки уникальности имени пользователя работает правильно.

5.3 Тестирование работы приложения при прерывании соединения с сервером

В данном тесте проверяется, как приложение реагирует на прерывание соединения с сервером во время работы. Это важно, поскольку в реальных условиях могут возникнуть ситуации, когда сервер становится недоступным во время сессии пользователя.

На представленном скриншоте (рисунок 5.1.3) видно, что при прерывании соединения с сервером, приложение выдает ошибку. Это

ожидаемое поведение, поскольку приложение не может продолжить работу без подключения к серверу.

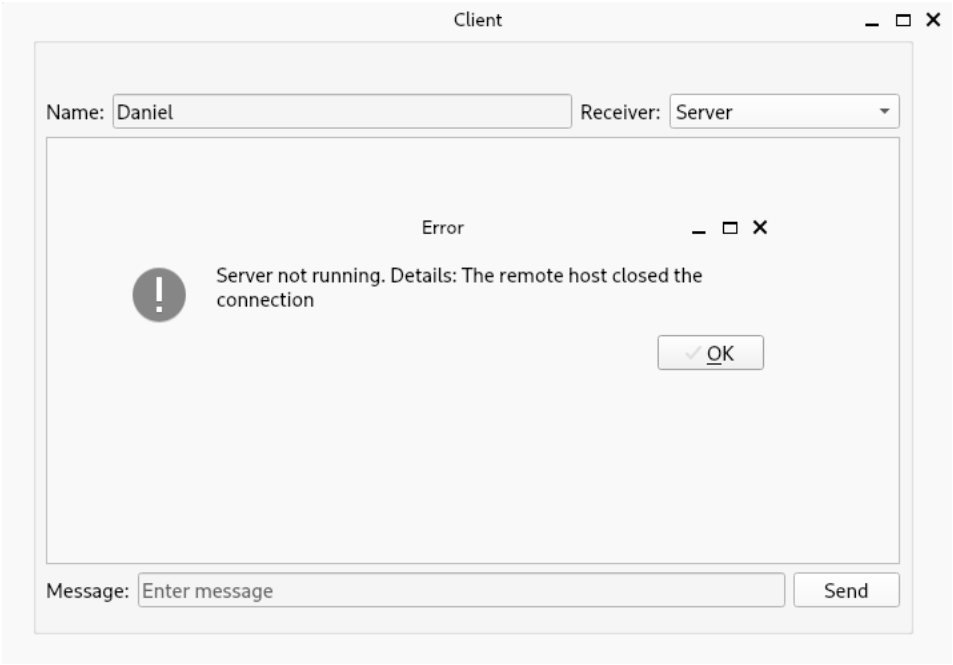

Рисунок 5.1.3 — Ошибка соединения при работе клиента в приложении

5.4 Тестирование функциональности отправки сообщений

В данном тесте проверяется, как приложение обрабатывает ситуацию, когда пользователь пытается отправить пустое сообщение.

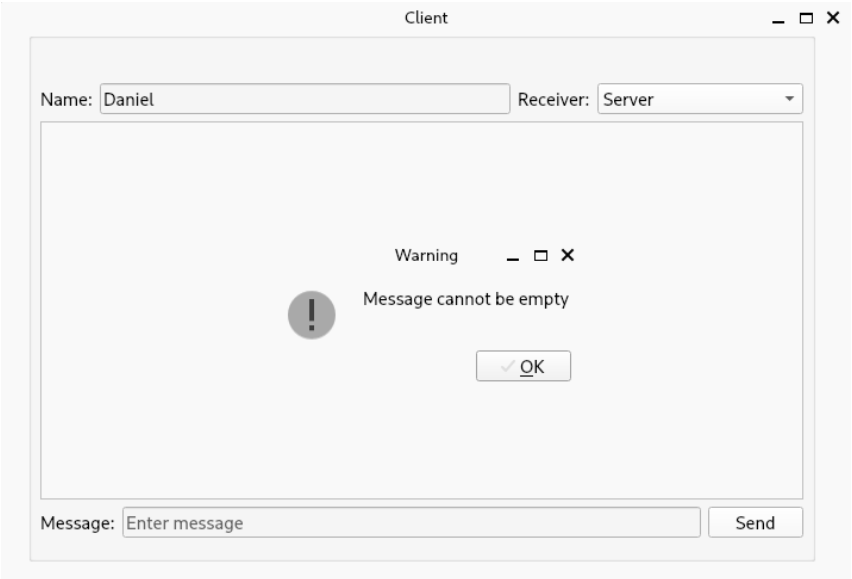

Рисунок 5.4.1 — Попытка отправки пустого сообщения

На представленном скриншоте (рисунок 5.4.1) видно, что при попытке отправить пустое сообщение, приложение корректно выдает предупреждение о том, что сообщение не может быть пустым.

5.5 Тестирование функциональности смены имени пользователя

В данном тесте проверяется, как приложение обрабатывает ситуацию, когда пользователь пытается сменить свое имя на имя, которое уже занято другим пользователем.

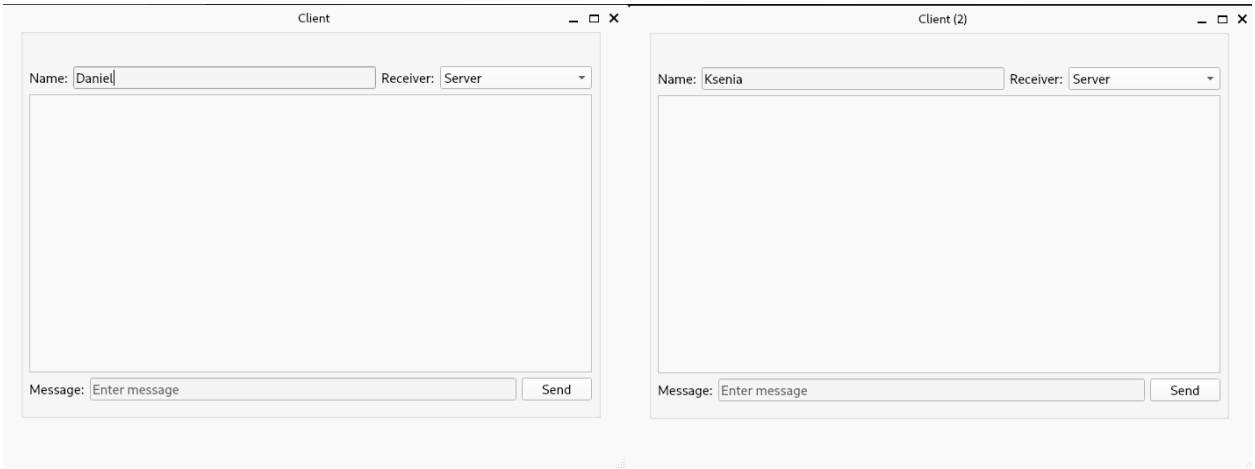

Рисунок 5.5.1 — Пользователи чата Daniel и Ksenia

На первом представленном скриншоте (рисунок 5.5.1) видно, что в чате уже присутствуют пользователи с именами "Daniel" и "Ksenia".

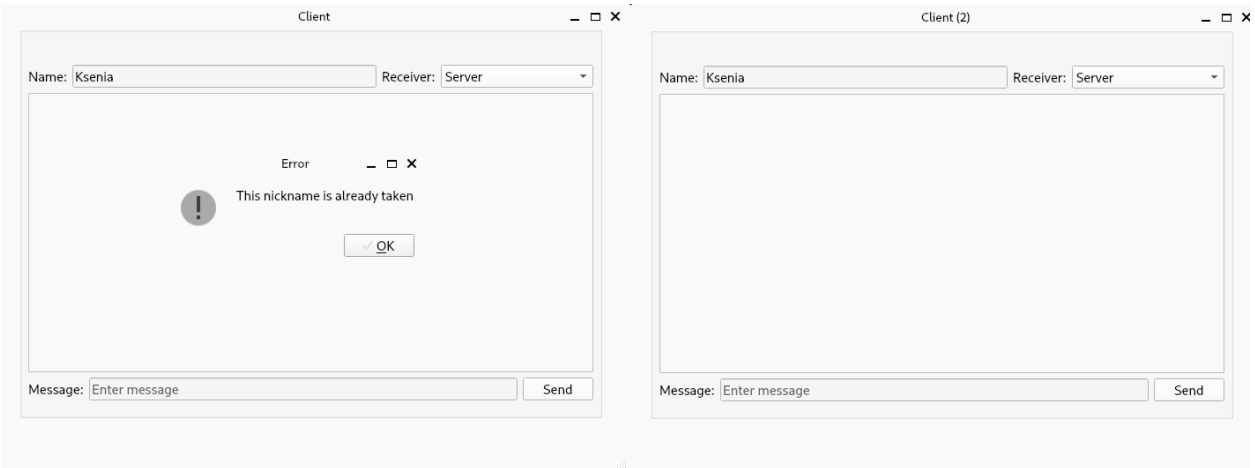

Рисунок 5.5.2 — Попытка смены имени на занятое.

На втором скриншоте (рисунок 5.5.2) видно, что пользователь "Daniel" пытается сменить свое имя на "Ksenia", но приложение корректно выдает предупреждение о том, что имя уже занято.

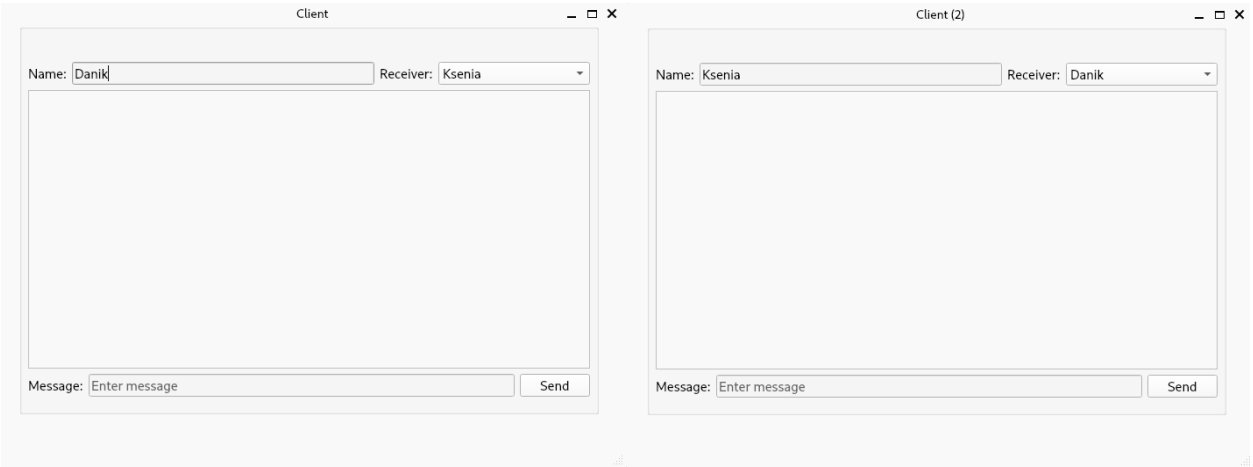

Рисунок 5.5.3 — Успешная смена имени пользователя.

На новом представленном скриншоте (рисунок 5.5.3) видно, что пользователь "Daniel" успешно сменил свое имя на "Danik". Пользователь "Ksenia" видит эту смену имени в своем интерфейсе чата.

5.6 Интерфейс сервера с управлением клиентами и отправкой сообщений

На сервере чат-приложения реализован интерфейс, который позволяет администратору видеть подключенных клиентов, отправлять им сообщения, а также отключать их при необходимости.

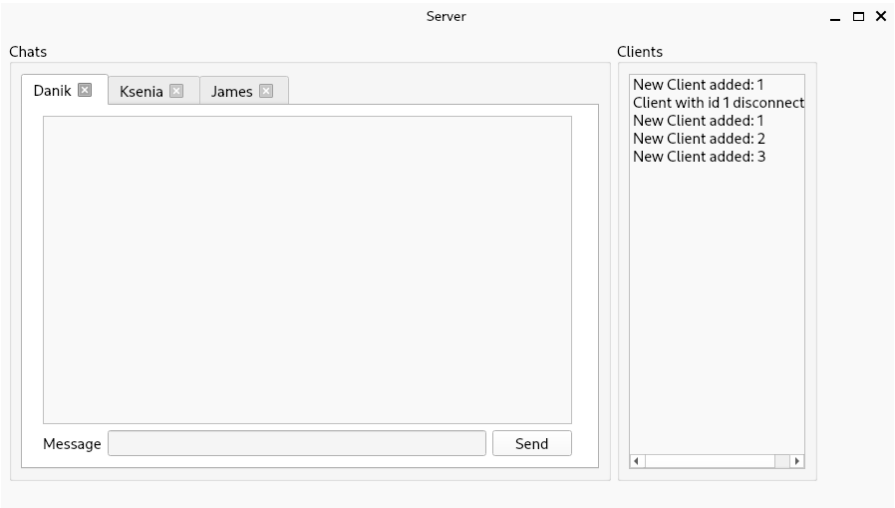

Рисунок 5.5.1 — Интерфейс сервера с управлением клиентами и отправкой сообщений.

На представленном скриншоте (рисунок 5.6.1) видно, как администратор сервера может видеть список подключенных клиентов, отправлять им сообщения и отключать их. Клиенты также могут отправлять сообщения на сервер, указывая тему сообщения.

Этот интерфейс позволяет администратору сервера активно участвовать в общении в чате и контролировать его, что может быть полезно для поддержания порядка и предотвращения злоупотреблений.

5.7 Тестирование функциональности обмена сообщениями между пользователями

В данном тесте проверяется, как приложение обрабатывает обмен сообщениями между пользователями. Это важно для поддержания коммуникации в чате.

На представленном скриншоте (рисунок 5.7.1) видно, что пользователь "James" отправляет сообщение пользователю "Ksenia".

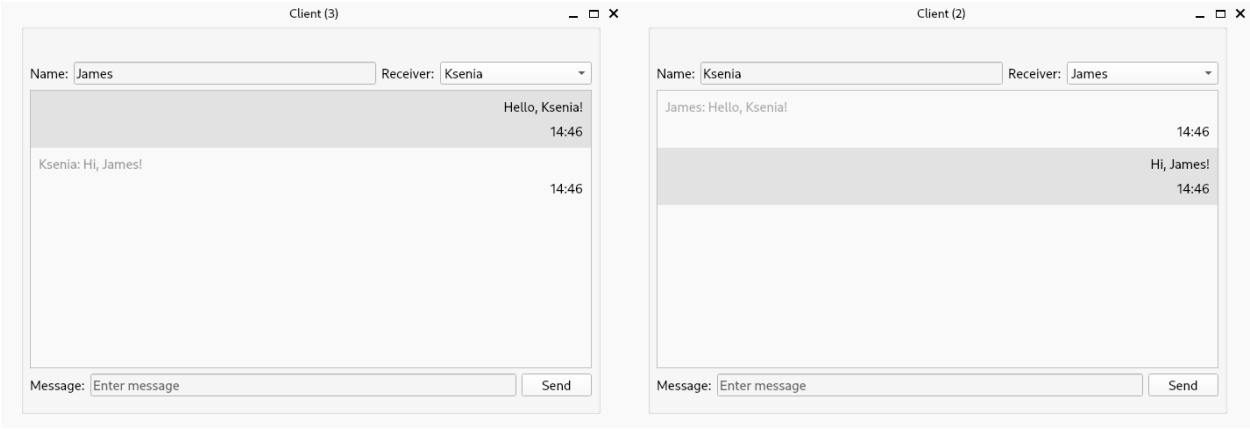

Рисунок 5.5.1 — Обмен сообщениями между James и Ksenia.

Также видно, что пользователь "Ksenia" получает сообщение от пользователя "James" и отвечает на него.

Это подтверждает, что функциональность обмена сообщениями между пользователями работает корректно.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска программы необходимо открыть файл «Client.exe». После этого откроется окно программы (рисунок 6.1).

Рисунок 6.1 — Окно программы при запуске.

Чтобы начать пользоваться приложением, введите имя пользователя и нажмите клавишу «Enter», после чего откроется рабочее окно приложения (рисунок 6.2).

Рисунок 6.2 — Окно программы после подключения.

В рабочем окне приложения вы увидите список подключенных пользователей (рисунок 6.3), а также принятые сообщения (рисунок 6.4).

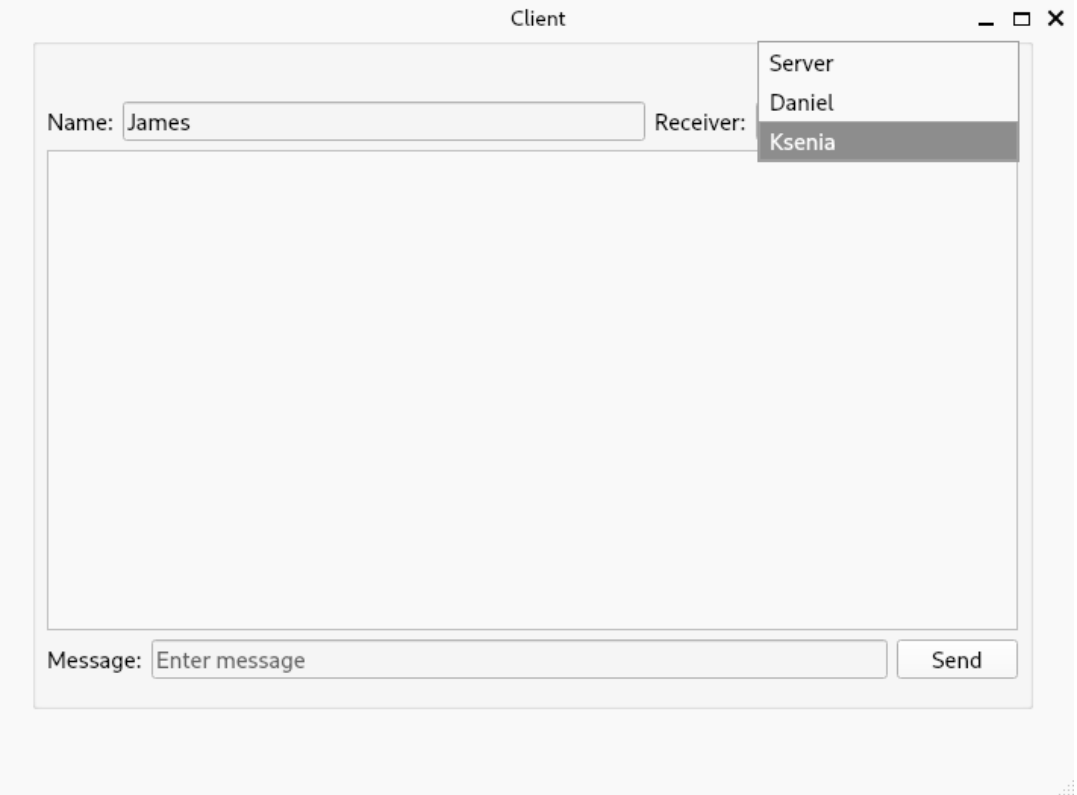

Рисунок 6.3 — Список подключенных пользователей.

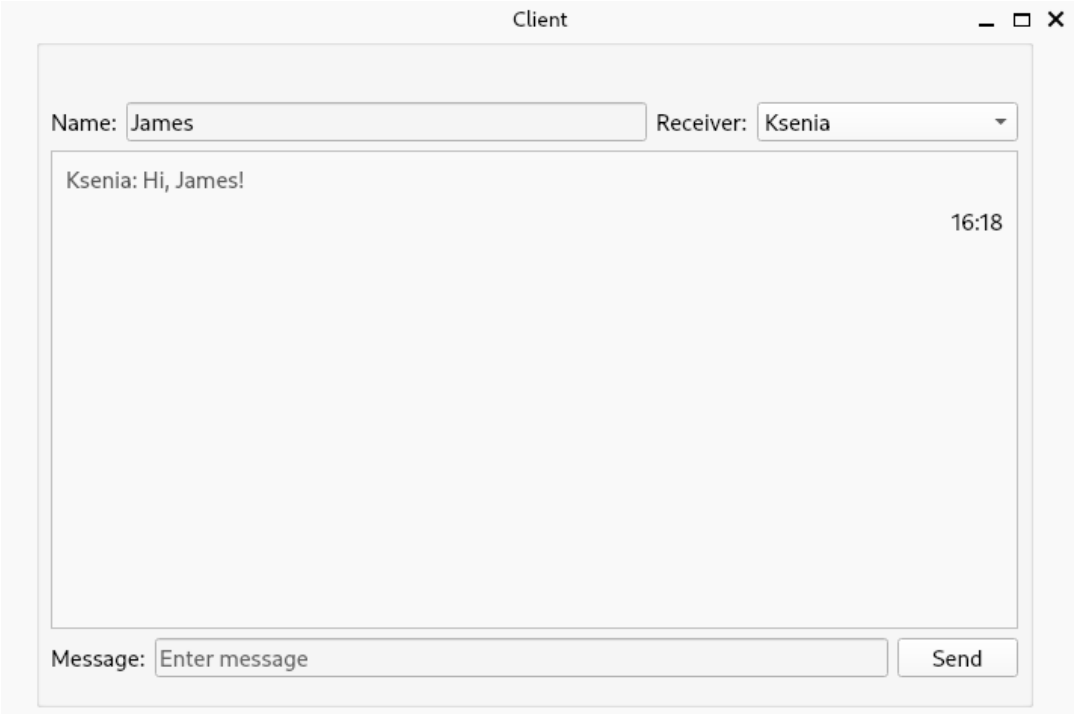

Рисунок 6.4 — Принятые и отправленные сообщения.

При подключении и отключении пользователей на экране выводится соответствующее сообщение, и список подключенных пользователей обновляется.

Чтобы написать сообщение в чат, введите текст для отправки или кнопку «Send» и нажмите Enter. Отправленные сообщения выравниваются по правому краю и задают сероватый фон, принятые от других пользователей сообщения и их имена выравниваются по левому краю. Это сделано для удобства чтения. Также при отправке и получении сообщения вы можете увидеть точное время его отправки.

Чтобы выйти из приложения, просто закройте его. После выхода программа завершится, освободится вся выделенная память, соединение будет разорвано, и окно исчезнет с экрана. Для повторного запуска просто снова запустите приложение.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта были успешно реализованы поставленные задачи, в результате чего были созданы клиентское и серверное приложения для обмена сообщениями. Были внедрены различные алгоритмы, включая обработку запросов клиента, обмен сообщениями между клиентами и управление подключениями со стороны сервера.

Клиентское приложение позволяет пользователям подключаться к серверу и обмениваться сообщениями с другими подключенными клиентами. Серверное приложение обеспечивает общение с каждым клиентом по отдельности и управляет всеми активными соединениями.

Приложения для обмена сообщениями, такие как разработанные в рамках этого проекта, являются отличным примером использования информационных систем для обеспечения эффективной коммуникации. Они предоставляют удобный и надежный способ обмена информацией, что может быть полезно в различных областях, от обучения до бизнеса.

В дальнейшем приложение может быть улучшено:

Шифрование отправляемых сообщений: это улучшит безопасность обмена сообщениями, защищая информацию от несанкционированного доступа.

Настройка интерфейса приложения: это позволит пользователям настраивать внешний вид и управление приложением, что улучшит пользовательский опыт.

Сохранение аккаунтов каждого пользователя с их историей: это позволит отслеживать историю общения каждого пользователя и улучшит управление информацией.

Добавление индикатора, если кто-то печатает: это улучшит интерактивность приложения, делая общение более динамичным и интересным.

ЛИТЕРАТУРА

[1] Страуструп, Б. Программирование. Принципы и практика использования C++ / Б. Страуструп. – М. : Вильямс, 2011. – 1225 с.

[2] Документация Qt [Электронный ресурс]. – Режим доступа: https://doc.qt.io/.

[3] Стандарты C++ [Электронный ресурс]. – Режим доступа: https://isocpp.org/std/the-standard.

[4] Вычислительные машины, системы и сети: дипломное проектирование [Электронный ресурс]. – Минск БГУИР 2019. - Режим доступа: https://www.bsuir.by/m/12_100229_1_136308.pdf.

[5] Лафоре, Р. Объектно-ориентированное программирование в C++, 4-е издание / Р. Лафоре – СПб.: Питер, 2004.

[6] Форум разработчиков на C++ [Электронный ресурс]. – Режим доступа: https://www.cplusplus.com/forum/.

[7] Шлее, М. Qt 5.10. Профессиональное программирование на C++ / М. Шлее. – СПб. : БХВ-Петербург, 2018. – 896 с.

[8] Форум разработчиков Qt [Электронный ресурс]. – Режим доступа: https://forum.qt.io/.

ПРИЛОЖЕНИЕ А

(обязательное)

Диаграмма классов сервера

ПРИЛОЖЕНИЕ Б

(обязательное)

Диаграмма классов клиента

ПРИЛОЖЕНИЕ В
(обязательное)
Схема структурная

ПРИЛОЖЕНИЕ Г
(обязательное)
Код программы